CSc 174 Database Management Systems

6. SQL: Assertions, Views, and Programming Techniques

Ying Jin
Computer Science Department
California state University, Sacramento

Constraints

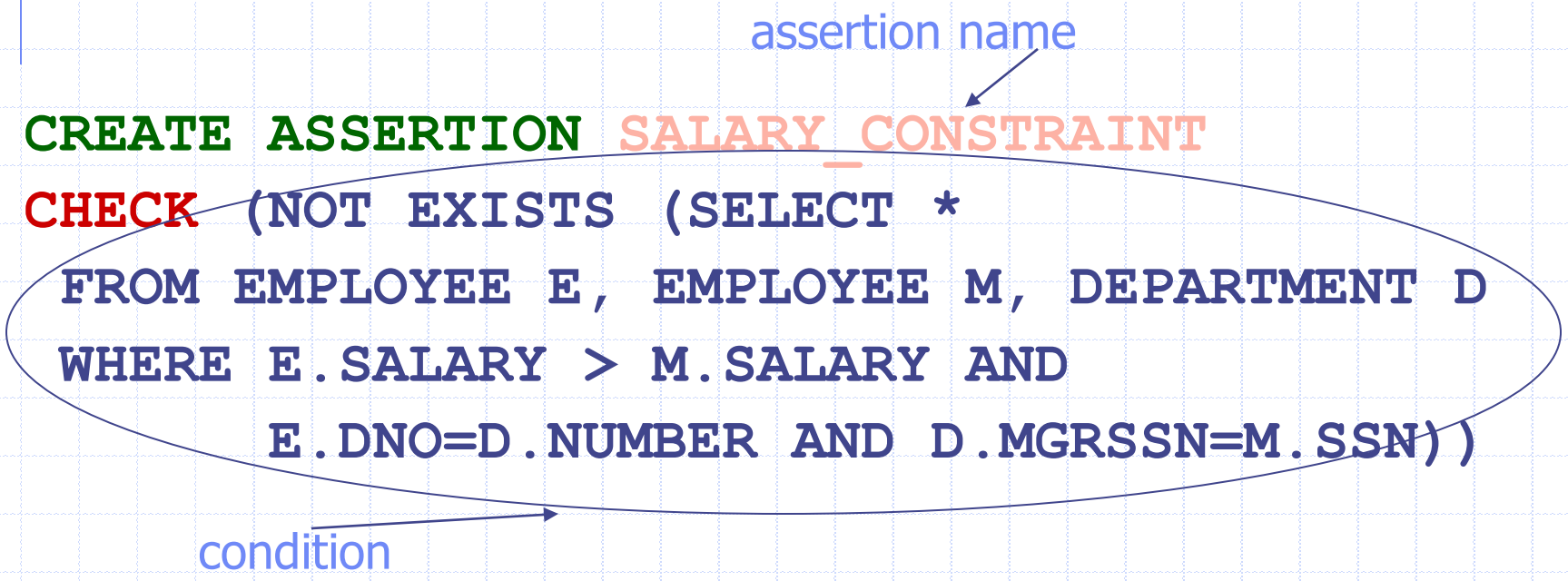
- Schema-based constraints
  - Domain constraints
  - Key constraints
  - Constraints on nulls
  - Referential integrity constraints
- Application-based constraints

Assertions

- **`CREATE ASSERTION`**
  - a constraint name
  - **`CHECK`**
  - a condition

Assertions: An Example

- The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.



Using General Assertions

- Condition
  - `NOT EXISTS` clause
  - Query
- Query result must be empty
  - Query result is not empty -> the assertion has been violated

SQL Triggers

- Violation
  - Assertion: Aborting an operation that causes a violation
  - Triggers: more options.
- Objective of triggers
  - Monitor a database and take action when a condition occurs

SQL Triggers (Cont.)

- Trigger syntax
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)
- Triggers can define references to *transition tables* that hold *old* and *new* values for the data that is updated as part of the change to the subject table.

MySQL Triggers: Example 1

- The trigger below will increase an employee's salary by 10% if the employee switches from department 01 to department 02.

MySQL Triggers: Example 2

NewEmp(Name, SSN, Salary, Dno)

NewDept(DName, DNO, Total_sal)

Total_sal: a derive attribute

- Use a trigger to maintain the derived attributes: when a new employee is added to the database, update the total salary coloumn in the NewDept table.

MySQL Triggers: Example 3

- Use a trigger to maintain the derived attributes: when an employee's gets updated (salary can be updated), update the total salary coloumn in the NewDept table.

Views in SQL

- "virtual" table
  - Derived from other tables
- Allows for limited update operations
- Allows full query operations

Specification of Views

- SQL command: `CREATE VIEW`
  - a table (view) name
  - a possible list of attribute names, e.g.:
    - Arithmetic operations are specified
    - We want the names to be different from the attributes in the base relations
  - a query
  - It is __?__responsibility to make sure view is always up to date.

SQL Views: An Example

- Specify a different WORKS_ON table

```
CREATE VIEW WORKS_ON_NEW AS
SELECT FNAME, LNAME, PNAME, HOURS
    FROM EMPLOYEE, PROJECT, WORKS_ON
    WHERE SSN=ESSN AND PNO=PNUMBER;
```

SQL Views Example: Specify Attribute Names

- **CREATE VIEW** DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)

AS SELECT DNAME, **COUNT**(*), **SUM**(SALARY)

FROM DEPARTMENT, EMPLOYEE

WHERE DNUMBER=DNO

GROUP BY DNAME;

- DEPT_INFO(DEPT_NAME, NO_OF_EMP,TOTAL_SAL)

Using a Virtual Table

- We can specify SQL queries on a newly create table (view):

```
SELECT FNAME, LNAME
FROM WORKS_ON_NEW
WHERE PNAME='Seena';
```

- When no longer needed, a view can be dropped:

```
DROP VIEW WORKS_ON_NEW;
```

Exercise

- (Query DEPT_INFO view) Get department name and the total salary of its employees, if this department has more than 5 employees.

Efficient View Implementation

- Option 1: Query modification
  - As a query on the underlying base tables
  - Disadvantage:
    - Inefficient for complex queries
    - Especially query views frequently within a short time period

Efficient View Implementation

- Option 2: View materialization
  - Physically creating and keeping a temporary table (within a certain time)
  - Assumption: other queries on the view will follow
  - Concerns: maintaining correspondence between the base table and the view when the base table is updated
  - Strategy: incremental update:
    - Update materialized view table when update base tables.

View Update

- Update on a single view without aggregate operations:
  - May map to an update on the underlying base table

Un-updatable Views

- Views defined using groups and aggregate functions are not updateable
- Views defined on multiple tables using joins are generally not updateable

Database Stored Procedures

- Persistent procedures/functions
- Advantages:
  - Needed by many applications?
    - -> Invoked by any of them
    - Reduce duplications
  - Allowing more complex types of derived data
  - Check for complex constraints that are beyond the specification power of assertion and triggers.

Stored Procedure Constructs

- A stored procedure

  CREATE PROCEDURE procedure-name (<params>)
  <local-declarations>
  <procedure-body>;

- A stored function

  CREATE FUNCTION <function name> (<params>)
  RETRUNS <return-type>
  <local-declarations>
  <function-body>;

- Calling a procedure or function (SQL standard)

  CALL <procedure-name or functionname> (<argument list>);

SQL Persistent Stored Modules (SQL/PSM)

- Part of the SQL standard
- Specify how to write persistent stored modules
- Includes additional programming constructs to enhance the power of SQL
  - e.g., branching and looping statements

SQL/PSM: An Example

```
CREATE FUNCTION DEPT_SIZE (IN deptno INTEGER)
RETURNS VARCHAR[7]
DECLARE TOT_EMPS INTEGER;
BEGIN
    SELECT COUNT (*) INTO TOT_EMPS
    FROM EMPLOYEE WHERE DNO = deptno;

    IF TOT_EMPS > 100 THEN RETURN "HUGE"
    ELSEIF TOT_EMPS > 50 THEN RETURN "LARGE"
    ELSEIF TOT_EMPS > 30 THEN RETURN "MEDIUM"
    ELSE RETURN "SMALL"
    ENDIF;

END;
```

MySQL stored procedure and functions

- Procedure
- Flow control structure
- Function examples

Procedure without parameters

- Example
  - List all employees
- To call a procedure: call list_all_emp()

Procedure with Input parameters

- Example
  - Retrieve the information of employee with the input SSN
  - call get_emp('123456789');

Procedure with output parameters

- Output parameter is indicated by OUT
- Example
  - Procedure
  - How to use the output parameter

Cursor

- Used inside procedure, functions, and triggers
- Not updatable
- Define variables first,
  Then define cursors,
  Then define handler.
- Handler is not associate with a specific cursor

```
delimiter $
CREATE PROCEDURE usecursor2()
BEGIN
  DECLARE no_more_row BOOLEAN DEFAULT FALSE;
  DECLARE fnameres VARCHAR(15);
  DECLARE lnameres VARCHAR(15);
  DECLARE cur1 CURSOR FOR SELECT FName, LName FROM Employee;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_row = TRUE;
  OPEN cur1;
  REPEAT
    FETCH cur1 INTO fnameres,lnameres;
    IF (no_more_row=FALSE) THEN    /* IF NOT no_more_row  THEN*/
       select fnameres,lnameres;
    END IF;
  UNTIL (no_more_row=TRUE) END REPEAT;
   /*UNTIL no_more_row END REPEAT;*/
  CLOSE cur1;
END $
delimiter ;
call usecursor2();
```

Flow Control Constructs

- Case
- Loop
- Repeat
- While

Case

- CASE *case_value*

  WHEN *when_value* THEN *statement_list* [WHEN *when_value* THEN *statement_list*] ... [ELSE *statement_list*]

  END CASE
- CASE

  WHEN *search_condition* THEN *statement_list* [WHEN *search_condition* THEN *statement_list*] ...

  [ELSE *statement_list*]

  END CASE

Loop

- Example
  - Define loop
  - LEAVE loop

Loop

```
delimiter $
CREATE PROCEDURE useloop()
BEGIN
  DECLARE no_more_row BOOLEAN DEFAULT FALSE;
  DECLARE fnameres VARCHAR(15);
  DECLARE lnameres VARCHAR(15);
  DECLARE cur1 CURSOR FOR SELECT FName, LName FROM Employee;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_row = TRUE;

  OPEN cur1;
  loop1: LOOP
    FETCH cur1 INTO fnameres,lnameres;
    IF no_more_row THEN
       CLOSE cur1;
       LEAVE loop1;
    END IF;
    select fnameres,lnameres;
  END LOOP loop1;

END $
delimiter ;
```

WHILE

- Example
  - Define while loop with conditions

WHILE

```
delimiter $
CREATE PROCEDURE usewhile()
BEGIN
  DECLARE no_more_row BOOLEAN DEFAULT FALSE;
  DECLARE fnameres VARCHAR(15);
  DECLARE lnameres VARCHAR(15);
  DECLARE cur1 CURSOR FOR SELECT FName, LName FROM Employee;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_row = TRUE;

  OPEN cur1;
  WHILE (NOT no_more_row) DO
     FETCH cur1 INTO fnameres,lnameres;
     IF NOT no_more_row THEN
       select fnameres,lnameres;
     END IF;
  END WHILE;
END $
delimiter ;
CALL usewhile();
```

Functions

- Example
  - Define a function
  - How to call a function

These slides are based on following textbook:

R. Elmaseri and S. Navathe, *Fundamentals of Database System*, 7th Edition, Addison-Wesley.

MySQL portion is based on the reference from:

MySQL online documentation: http://dev.mysql.com